Repaso para prueba objetiva 2
Concurrencia
2010-2011 - Segundo semestre
Lenguajes, Sistemas Informáticos e Ingeniería de Software

(2 puntos) 1. Dado el siguiente **CTAD** (sólo se muestran las partes necesarias):

TIPO: Factoría $= \acute{A}rea \nrightarrow \mathbb{N}$
DONDE: $\acute{A}rea = \{0, 1\}$
INVARIANTE: $\forall f \in$ *Factoría*. $(\forall\, a \in \acute{A}rea.\, f(a) \leq 10) \wedge \sum_{a \in \acute{A}rea} f(a) < 20$

INICIAL(f)**:** $\sum_{a \in \acute{A}rea} f(a) = 0$

CPRE: $f(a) < 10 \wedge \sum_{a \in \acute{A}rea} f(a) < 19$
Entrar(f,a)
POST: $f = f^{pre} \oplus \{a \mapsto f^{pre}(a) + 1\}$

CPRE: Cierto
Salir(f,a)
POST: $f = f^{pre} \oplus \{a \mapsto f^{pre}(a) - 1\}$

Suponemos un programa concurrente en el que los procesos respetan el siguiente protocolo (o esquemas de llamada): *Entrar*(f, a); ...; *Salir*(f, a).

A continuación se muestran las partes relevantes (cliente y servidor) de una implementación del recurso utilizando paso de mensajes.

```
private Any2OneChannel chEntrar = Channel.any2one();
```

```
// Ejecutado por el cliente
public void entrar(int a) {
  One2OneChannel sincro =
    Channel.one2one();
  Object[] pet =
    {new Integer(a),sincro};
  chEntrar.out().write(pet);
  sincro.out().write(null);
}
```

```
// Entrada en la select del servidor
case ENTRAR:
  Object[] pet =
    (Object[])chEntrar.in().read();
  int a = ((Integer)pet[0]).intValue();
  if (dentro[a] < 10 && total < 19) {
    dentro[a]++; total++;
    ((One2OneChannel)pet[1]).in().read();
  }
  else
    esperanEntrar.enqueue(pet);
  break;
```

Como se puede observar se ha modificado el esquema metodológico seguido en clase con *send-receive* en el cliente y *receive-send* en el servidor por esquemas *send-send* y *receive-receive* respectivamente.

Se pide señalar la respuesta correcta.

(a) Con dicho cambio el código no compilaría.

(b) Para este ejemplo el esquema propuesto es igualmente válido.

(c) Para este ejemplo el esquema propuesto es incorrecto.

(d) Ninguna de las otras respuestas es correcta.

(2 puntos) 2. Dado el recurso del ejercicio anterior, mostramos a continuación otra implementación usando paso de mensajes:

```
// Canales de comunicación
private Any2OneChannel chEntrar = Channel.any2one();
private Any2OneChannelInt chSalir = Channel.any2oneInt();
```

```
// Ejecutado por el cliente
public void entrar(int a) {
  One2OneChannel sincro =
    Channel.one2one();
  Object[] pet =
    {new Integer(a),sincro};
  chEntrar.out().write(pet);
  sincro.in().read();
}

public void salir(int a) {
  chSalir.out().write(a);
}

//  Código del servidor
public void run() {
  // Estado del recuros compartido
  int dentro[] = {0, 0};
  int total = 0;

  // Cola de bloqueados
  Queue<Object[]> esperanEntrar =
    new NodeQueue<Object[]>();

  // Preparando la recepción
  // no determinista
  final int ENTRAR = 0;
  final int SALIR = 1;
  Guard[] entradas =
    {chEntrar.in(), chSalir.in()};
  Alternative servicios =
    new Alternative(entradas);
  boolean[] sincCond =
    new boolean[2];

  // Variables auxiliares
  int a;
  Object[] pet;
  One2OneChannel chResp;
```

```
  // Bucle principal del servidor
  while (true) {
    sincCond[ENTRAR] = total < 19;
    sincCond[SALIR] = true;

    // Aceptamos y almacenamos peticiones
    switch (servicios.fairSelect(sincCond)) {
    case ENTRAR:
      pet = (Object[])chEntrar.in().read();
      a = ((Integer)pet[0]).intValue();
      chResp = ((One2OneChannel)pet[1]);
      if (dentro[a] < 10) {
        dentro[a]++; total++;
        chResp.out().write(null);
      }
      else
        esperanEntrar.enqueue(pet);
      break;
    case SALIR:
      a = chSalir.in().read();
      dentro[a]--;  total--;
      break;
    default:
      break;
    }
    int n = esperanEntrar.size();
    for (int i = 0; i < n; i++) {
      pet = esperanEntrar.front();
      a = ((Integer)pet[0]).intValue();
      chResp = ((One2OneChannel)pet[1]);
      if (dentro[a] < 10) {
        dentro[a]++; total++;
        chResp.out().write(null);
      }
      else {
        esperanEntrar.dequeue();
        esperanEntrar.enqueue(pet);
      }
    }
  }
}
```

Se pide señalar la respuesta correcta.

(a) La expresión `chResp.out().write(null)` no compila.

(b) Es una implementación correcta del recurso compartido.

(c) La operación `entrar` puede atenderse sin que se cumpla su *CPRE*.

(d) Pueden quedar procesos bloqueados en `esperanEntrar` a pesar de que su *CPRE* se cumple.

(Realmente, ya se ha comentado en clase que la opción correcta es la (c). Se trata de encontrar un escenario que provoque el fallo.)